SWIP at QALD-3: results, criticisms and lesson learned

Camille Pradel, Guillaume Peyet, Ollivier Haemmerlé, and Nathalie Hernandez

IRIT, Université de Toulouse le Mirail, Département de
Mathématiques-Informatique, 5 allées Antonio Machado, F-31058 Toulouse Cedex
{camille.pradel, guillaume.peyet, ollivier.haemmerle,
nathalie.hernandez}@univ-tlse2.fr

Abstract. This paper presents the results obtained by the SWIP system while participating in the QALD-3 (Question Answering over Linked Data) challenge, co-located with CLEF 2013 (Conference and Labs of the Evaluation Forum). We tackled task 1, multilingual question answering, whose purpose is to interpret natural language questions in order to return the answers contained in a graph knowledge base. We answered queries of both proposed datasets (one concerning DBpedia, the other Musicbrainz) and took into consideration only questions in English.
The system SWIP (*Semantic Web Interface using Patterns*) aims at automatically generating formal queries from user queries expressed in natural language. For this, it relies on the use of query patterns which enable the complex task of interpreting natural language queries.
The results obtained on the Musicbrainz dataset (precision = 0.51, recall = 0.51, F-measure = 0.51) are very satisfactory and encouraging. The results on DBpedia (precision = 0.16, recall = 0.15, F-measure = 0.16) are more disappointing.
In this paper, we present both the SWIP approach and its implementation. We then present the results of the challenge in more detail and their analysis. Finally we draw some conclusions on the strengths and weaknesses of our approach, and suggest ways to improve its performance.

1 Introduction

End-users need to access the huge amount of data available over the Internet. Databases are hidden by means of forms which lead generally to SQL queries. With the increasing publication of linked data, a need for interfacing SPARQL engines emerged, since it is impossible for an end-user to handle the complexity of the "schemata" of these pieces of knowledge.

Our work is situated in this field of research: how can we interpret a natural language query and translate it into SPARQL? Our main postulate states that, in real applications, the submitted queries are variations of a few typical query families. Our approach differs from existing ones in that we propose to guide the interpretation process by using predefined query patterns which represent these query families. The use of patterns avoids exploring the ontology to link

the semantic entities identified from the keywords since potential query shapes are already expressed in the patterns. The process thus benefits from the pre-established families of frequently expressed queries for which we know that real information needs exist.

This article presents and analyses the results obtained by the SWIP system for task 1 of the QALD-3 challenge[1]. The purpose of this challenge is to evaluate systems on their ability to retrieve answers to a set of natural language queries from an RDF dataset. Participants were given a training dataset (a set of questions with their SPARQL translations) in order to adapt and tune their systems, and then had to provide the generated interpretations of NL queries from the test set.

After presenting the main approaches aiming at the same objectives as SWIP in Section 2, we give an overview of our approach in Section 3. Then, in Section 4, we present and analyse the results of the challenge, before concluding and mentioning potential future improvements to our system in Section 5.

2 Existing approaches

In this section, we present approaches from the literature aiming at helping users to query graph based knowledge bases. For this, we stick to the classification proposed in [8], and present them following the formality continuum, from most formal to most permissive, beginning with the query languages themselves.

On an extreme side of this continuum are the formal graph languages, such as SPARQL[2]. They are the targets of the systems we are presenting in this section. Such languages present obvious usability constraints, which make them unsuited to end-users. Expressing formal queries implies knowing and respecting the language syntax used, understanding a graph model and, most constraining, knowing the data schema of the queried knowledge base. The work presented in [5] aims at extending the SPARQL language and its querying mechanism in order to take into account keywords and wildcards when the user does not know exactly the schema he/she wants to query on. Here again, such an approach requires that the user knows the SPARQL language.

Similar are approaches assisting the user during the formulation of queries in such languages. Very light interfaces such as Flint[3] and SparQLed[4] implement simple features such as syntactical coloration and autocompletion. Other approaches rely on graphical interfaces such as [1] for RQL queries, [15,3] for SPARQL queries or [4] for conceptual graphs. Even if these graphical interfaces are useful and make the query formulation work less tedious, we believe they are not well suited to end-users since they do not overcome the previously mentioned usability limits of formal graph query languages.

[1] `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=3`
[2] http://www.w3.org/TR/rdf-sparql-query/
[3] http://openuplabs.tso.co.uk/demos/sparqleditor
[4] http://sindicetech.com/sindice-suite/sparqled/

Sewelis [6] introduces *Query-based Faceted Search*, a new paradigm combining faceted search and querying, the most popular paradigm in semantic data search, while other approaches such as squall2sparql [7] define a controlled natural language whose translation into SPARQL is straightforward.

Other works aim at generating automatically – or semi-automatically – formal queries from user queries expressed in terms of keywords or NL. Our work is situated in this family of approaches. The user expresses his/her information need in an intuitive way, without having to know the query language or the knowledge representation formalism used by the system. Some works have already been proposed to express formal queries in different languages such as SeREQL [10] or SPARQL [19,16,2]. In these systems, the generation of the query requires the following steps: (i) matching the keywords to semantic entities defined in the KB, (ii) building query graphs linking the entities previously detected by exploring the KB, (iii) ranking the built queries, (iv) making the user select the right one. The existing approaches focus on several main issues: optimizing the first step by using external resources (such as WordNet or Wikipedia)[10,18], optimizing the knowledge exploration mechanism for building the query graphs [19,16], enhancing the query ranking score [18], and improving the identification of relations using textual patterns [2].

Autosparql [9] extends the previous category: after a basic interpretation of the user NL query, the system interacts with the user, asking for positive and negative examples (i.e. elements which are or are not in the list of expected answers), in order to refine the initial interpretation by performing a learning algorithm. In [17] the interpretation process of this same system is improved by determining a SPARQL template from the syntactic structure of the natural language question.

3 Overview of the SWIP approach

In the SWIP system, the query interpretation process consists of two main steps: the translation of the NL user query into a pivot query, and the formalization of this pivot query, respectively described in subsections 3.1 and 3.2. The first step roughly consists in the identification of the query focus, a dependency analysis and the generation of a new query, called pivot query, which explicitly represents extracted relations between substrings of the NL query sentence. In the second step, predefined query patterns are mapped to the pivot query; we thus obtain a list of potential interpretations of the user query, which are then ranked according to their estimated relevance and proposed to the user in the form of reformulated NL queries.

3.1 From natural language to pivot query

The user query is translated into a new structure called the *pivot query*. This structure is half way between the NL query and the targeted formal query, and can be expressed through a language, called pivot language, which is formally

defined in [13]. We use this pivot language in order to facilitate the implementation of multilingualism by means of a common intermediate format: a specific module of translation of NL to pivot has to be written for each different language, but the pivot query formalization step remains unchanged. The translation of a natural language query into a pivot query is based on the use of a syntactic dependency parser which produces a graph where nodes correspond to the words of the sentence and edges to the grammatical dependencies between them.

Named entity identification Before parsing the query, a first stage identifies in the sentence the entities corresponding to knowledge base resources. These entities are then considered as a whole and will not be separated by the parser in the next stage. For example, in the sentence "who are the actors of Underworld: Awakening", "Underworld: Awakening" will be considered as a named entity as it is the label of an instance of Film in the knowledge base. This stage is particularly crucial when querying knowledge bases containing long labels, such as group names or film titles are that made up of several words or even sometimes of a full sentence. Once the resource entities are identified and the dependency graph is generated, a set of rules are applied to construct the pivot query.

Query focus identification First, the focus of the query is searched for in order to identify the specific information the user is interested in. To do so, the graph is browsed looking for an interrogative word which will help identify the query focus either by the nouns linked to it in the graph (for "Which films did Ted Hope produce?", the focus will be "film") or by the type of answer it expects (for "When did Charles Chaplin die?", the focus will be "?date"). If no interrogative word is found, we look for specific phrases ("List", "What are", ...) introducing list queries, which are queries that also expect a list of resources fulfilling certain conditions. In this case, the focus of the query will be the lemma of the noun identified as the direct object of the verb in the clause in which the phrase is found (for "List all the films in which Charles Chaplin acted", the focus will be "?film"). If, after this process, no focus has been identified, we consider the query as dichotomous which means it will allow only two answers: True or False (for "Was Jean Dujardin involved in the film The Artist?": there will be no focus defined).

Subquery generation The dependency graph is once more browsed in order to identify from the different clauses of the sentence the possible subqueries constituting the pivot query, i.e. the elements of the query and their relations.

The following rules are applied. If two nodes are part of a nominal phrase, a binary subquery is constructed according to the lemma of the head and the lemma of the expansion. For example, for the query "Give me all the films of Jean Dujardin", the generated binary query will be `?"film": "Jean Dujardin"`. If three nodes are part of a clause composed of a subject, a verb and an object, a ternary subquery is constructed according to the lemma of each node. For the

clause “Who played in the Artist?”, the corresponding ternary subquery will be `?"person": "play"= "the Artist"`. If the part of speech of a node is a relative pronoun, the keyword used for representing this node in a subquery will be the lemma of the noun it references. For example, for the query “Who played in a film in which Jean Dujardin also played?”, the subquery corresponding to the second clause will be `"film": "play"= "Jean Dujardin"`, the subquery generated for the first clause is `?"person": "play"= "film"`. Note that the use of the same keyword allows us to express that the same film has to be considered in both subqueries.

For each node, each rule is analyzed. This means that several subqueries can be generated from one clause. For example, for the clause“Was Jean Dujardin involved in the film The Artist?”, both subqueries `?"film": "The Artist"` and `"Jean Dujardin": "involved"= "The Artist"` will be generated.

These rules might seem simple but we have observed that the structure of queries expressed by end-users is generally simple.

3.2 From pivot to formal query

Formalizing pivot queries using query patterns was the first task we tackled and is thus extensively described in [12] and [13]. We briefly describe here the structure of a query pattern and the process of this formalization.

A pattern is composed of an RDF graph which is the prototype of a relevant family of queries. Such a pattern is characterized by a subset of its elements – either class, property or literal type –, called the qualifying elements, which can be modified during the construction of the final query graph. It is also described by a sentence in natural language in which a distinct substring must be associated with each qualifying element. For now, the patterns are designed by experts who know the application domain. The designer of a pattern builds its RDF graph manually, selects its qualifying elements and gives the describing sentence.

The formalization of the pivot query is processed as follows. Each element of the user query expressed in the pivot language is matched to an element of the knowledge base. Elements of the knowledge base can either be a class, a property, an instance or a literal type (which can be any type supported by SPARQL, i.e. any type defined in RDF Schema). Then we map query patterns to these elements. The different mappings are presented to the user by means of natural language sentences. The selected sentence allows the final SPARQL query to be built.

A recent evolution of the pattern structure makes the patterns more modular and the query generation more dynamic. We can now assign values of minimal and maximal cardinalities to subgraphs of the patterns, making these subgraphs optional or repeatable when generating the formal query. The descriptive sentence presented to the user also benefits from this novelty and no longer contains non relevant parts (parts of the pattern which were not addressed by the user query), thus making our system more ergonomic.

Sentence template: [c_1] ***if_begin(****was formed on/in* [l_1]*,) **if_end(****broke up on/in* [l_2]*).*

Fig. 1. Examples of complex patterns, containing optional or repeatable subpatterns.

3.3 Example

Figure 1 shows an example of pattern which was built for the Musicbrainz dataset. In this example and in the rest of this paper, we express URIs with prefixes without making them explicit; they are common prefixes and must be considered with their usual value. It covers all questions about creation and dissolution dates of music groups, and can be used to interpret, amongst others, query 33 from the Musicbrainz test query set, "When were the Dixie Chicks founded?" The pivot translation obtained by the Swip system for this question is `found: Dixie_Chicks= ?date`. By mapping qualifying vertex $c1$ to keyword `Dixie_Chicks`, $c2$ to `found` and $l1$ to `date`, and omitting the subpattern *end*, the Swip system generates the SPARQL interpretation presented in Figure 2.

```
SELECT DISTINCT ?date WHERE
{
  ?Dixie_Chicks rdfs:label "Dixie Chicks".
  ?found bio:date ?date.
  ?Dixie_Chicks bio:event ?found.
  ?found rdf:type bio:Birth.
  ?Dixie_Chicks rdf:type mo:MusicGroup.
}
```

Fig. 2. SPARQL interpretation returned by Swip to query 33 "When were the Dixie Chicks founded?"

4 Evaluation on QALD-3 datasets

A prototype of our approach was implemented in order to evaluate its effectiveness. It is available at `http://swip.univ-tlse2.fr/SwipWebClient`. It was implemented in Java and uses the MaltParser [11] for the dependency analysis of English user queries. The system performs the second main process step (translating from pivot to formal query) by exploiting a SPARQL server based

on the ARQ[5] query processor, here configured to exploit LARQ[6], allowing the use of Apache Lucene[7] features, such as indexation and Lucene score (used to obtain the similarity score between strings).

Experiments were carried out on the evaluation framework proposed in task 1 of the QALD-3 challenge[8]. The evaluation method was defined by the challenge organizers. It consists in calculating, for each test query, the precision, the recall and the F-measure of the SPARQL translation returned by the system, compared with handmade queries of a gold standard document. We participated in both subtasks proposed by the challenge organizers, one targeting the DBpedia[9] knowledge base and the other targeting an RDF export of Musicbrainz[10] based on the music ontology [14]. The quality of the results varies with the target KB; these results are presented in this section.

4.1 Results overview

The Musicbrainz test dataset was composed of 50 NL questions. We processed 33 of them. 24 were correctly interpreted, 2 were partially answered and the others failed. The average precision, recall and F-measure, calculated by the challenge organizers, are all equal to 0.51. It is difficult to evaluate these performances, as no other challenge participant tackled the Musicbrainz dataset. However, these results are quite good when compared to those obtained by participants on the DBpedia dataset. Indeed, among the other participants, the squall2sparql system showed the best performance by far, but this system does not accept full NL sentences as an input [7], and then, the second best F-measure is 0.36. Of course the relevance of such a direct comparison is very questionable, as both graph bases present significant differences and our system obtained poor results on DBpedia, for reasons that are explained later.

The results obtained by SWIP on the DBpedia dataset, composed of 100 test questions, are not as good as our results on Musicbrainz. We processed 21 questions, of which 14 were successful, 2 were partially answered and 5 were not correct. The precision (0.16), the F-measure (0.16) and the recall (0.16) are quite low.

4.2 Evolution since QALD-1

These results obtained by SWIP can be compared to those we obtained two years ago, when participating in the first edition of QALD challenge[11]. Here again, we

[5] http://openjena.org/ARQ/

[6] LARQ = Lucene + ARQ, see http://jena.sourceforge.net/ARQ/lucene-arq.html

[7] http://lucene.apache.org/

[8] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=3

[9] http://dbpedia.org

[10] http://musicbrainz.org/

[11] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=1

cannot make to a direct figure comparison as the evaluation method was slightly different from the one applied in QALD-3, and the RDF export of Musicbrainz was not based on the Music ontology as it is now, but rather on a schema which was basically the translation of the relational schema used by the Musicbrainz database. The new evaluation measures, when applied to our QALD-1 results, gave us an average precision (resp. recall, F-measure) of 0.59 (resp. 0.59, 0.58). The new results could seem lower than the previous ones, but we have to take into account that the translation step from NL to pivot queries described in 3.1 is now automated, while it was carried out manually for QALD-1. As the task of natural language interpretation is very complex, we consider these new results as very encouraging.

4.3 Pattern construction and coverage

The Musicbrainz's query patterns used for this evaluation were built by adapting (to fit the new ontology) and updating (to take into account the 50 new training queries) the ones created for QALD-1. We thus obtained five query patterns. The one presented in Figure 1 is the simplest of them. The others can be visualized through the user interface. Among the set of 50 test questions, only two required query graphs which could not be generated by one of the patterns, without taking into account the "out of scope" queries whose answer is not in the KB anyway. This fairly high coverage rate shows that our approach is suitable for closed domain KB, whose data is consistent with the schema.

For DBpedia, the first patterns were implemented with the QALD training questions. Without taking into account the "out of scope" questions, we have implemented all questions in patterns; but when the QALD test questions were submitted, we noticed that only 19% of them were covered by the training patterns. In comparison, Musicbrainz patterns covered 96% of the test questions.

The main problem for us was the "tidiness" of the processed data. Indeed, most of the DBpedia knowledge base is automatically generated from information written by thousands of contributors and not exported from a consistent database like Musicbrainz. Consequently, we had to deal with the heterogeneity and inconsistency of the target KB that SWIP was not able to overcome. For instance, the nickname of an entity can be expressed in the KB by four distinct properties (`dbo:nickname`, `dbo:nicknames`, `dbp:nickname` and `dbp:nicknames`). As another example, consider the question "Which states of Germany are governed by the Social Democratic Party?" Its SPARQL translation is presented in figure 3, the value of the property `dbp:rulingParty` can be either a URI referring to the Social Democratic Party, or an RDF plain literal.

We can deduce that our pattern based approach is very well suited for "clean" KB, with consistent data, domain specific and respecting the schema it is based on. However, in its current implementation, it is not well suited to KBs containing many inconsistencies, which are the most popular KB on the Web of linked data.

```
SELECT DISTINCT ?uri
WHERE {
  ?uri rdf:type yago:StatesOfGermany .
  { ?uri dbp:rulingParty 'SPD'@en. }
   UNION
  { ?uri dbp:rulingParty res:Social_Democratic_Party_of_Germany. }
}
```

Fig. 3. Gold standard SPARQL translation of question 8 from DBpedia's training dataset.

4.4 Unsupported queries

As explained above, 17 queries about Musicbrainz and 81 about DBpedia were not processed by the SWIP system. We identified some categories whose expressivity exceeds that currently handled by our approach. Most of the non-supported queries contain aggregates, like "Which bands recorded more than 50 albums?" For now, the only aggregate function supported by SWIP is the counting of the total number of results of a query, as in "How many singles did the Scorpions release?" because it is easy to spot in a NL query and it seems to be the most recurrent in user queries. Furthermore, SWIP is not able to handle queries which require inferences made beforehand, such as "Which artists turned 60 on May 15, 2011?", and queries implying string comparison, such as "Give me all bands whose name starts with Play." Of course, adding the support of these categories would significantly improve our approach's results in the challenge, and the design of these new features is the natural next step for our future work, starting with the aggregates which seem to us to be the most recurrent in users' needs.

4.5 Failed queries

We here give details about some queries whose interpretation failed. We explain why they failed and emphasize some flaws in the current implementation of SWIP and also some imperfections in the challenge dataset.

Musicbrainz dataset In its SPARQL translation of queries 15 "Who composed the song Coast to Coast?" and 48 "Give me all soundtracks composed by the Pet Shop Boys.", SWIP made use of the property `mo:composer` which seems to translate the NL query's information need more appropriately than the gold standard query which uses `foaf:maker`. However, the generated SPARQL queries did not return any result and were thus considered wrong.

Another case of failed queries which were actually correctly interpreted are queries 19 "With whom did Phil Collins work together?", 50 "On which singles did Robbie Williams collaborate with Nicole Kidman?" and 51 "Did Kylie Minogue ever collaborate with Mariah Carey?" These queries imply parts of

the KB which are not consistent with the involved vocabulary (Relationship[12]). Indeed, the Musicbrainz KB (and thus, each gold standard query) misuses the property rel:collaboratesWith which, according to the Relationship vocabulary, is not supposed to be used with an intermediary blank node (symmetrical property with domain and range foaf:Person).

For question 46 "Give me the titles of all singles by Phil Collins.", SWIP returned the resources themselves instead of their titles. Indeed, SWIP ignores all references to labels (or any lexical representation, like titles or names) for two main reasons:

- we believe that the values of these properties are naturally used in NL without specifying the nature of the property (we usually say "Michael Jackson" and not "somebody whose name is Michael Jackson");
- the user interface uses labels to show results to the user anyway (is this not what labels are for?).

Questions 32 "How many pieces of work did Mozart create?" and 39 "Does the song Peggy Sue appear on Buddy Holly's Greatest Hits compilation?" failed because of the inaccuracy of the matching step results. In query 32, SWIP matched the keyword "Mozart" with the resource with this same word as the label and ignored the label "Wolfgang Amadeus Mozart". In query 39, the whole string "Buddy Holly's Greatest Hits" was recognized as a named entity and was matched to the resource with the same value as a label, while the gold standard query makes use of a resource with the label "Greatest Hits" which has for `foaf:maker` another resource with the label "Buddy Holly". To overcome these ambiguities, work should be done to improve this step which should be more flexible and maybe take into account the popularity of resources (thus a great and very famous composer could be favored to an obscure music group).

Question 11, "How long is Louder Than Words by Against All Authority?", failed because of a simple typo in the concerned pattern. Indeed, the pattern used the property `duration` of Music ontology, instead of that of the timeline ontology which is actually used in the KB. This error was introduced while adapting the patterns to the new ontology. The question was actually correctly interpreted but the generated SPARQL made use of a non-existing property. This kind of error could be easily prevented by using a simple tool allowing to edit patterns and check that they fulfill some simple conditions.

Finally, the interpretation of question 5 "How many tracks does Erotica have?" was considered as correct according to the gold standard document, but actually neither the generated SPARQL query, nor the gold standard one is correct. Indeed, the gold standard translation of question 5 showed in figure 4 returns 40, the total number of tracks in all albums named "Erotica". In this case, several instances of the same album by Madonna are considered and each track is thus taken into account several times. The tracks of another album named "Erotica" by Roberto Perera are taken into account. All these problems can be illustrated by the query showed in figure 5. This query returns (logically)

[12] `http://vocab.org/relationship/collaboratesWith.html`

40 results, involving 40 distinct URIs for the variable `track`, but we can see that some of these tracks have the same title and do not belong to the same album.

```
SELECT COUNT(DISTINCT ?track)
WHERE {
  ?album dc:title 'Erotica' .
  ?album mo:track ?track .
}
```

Fig. 4. Gold standard SPARQL translation of question 5 from Musicbrainz test dataset.

```
SELECT *
WHERE {
  ?album dc:title 'Erotica' .
  ?album mo:track ?track .
  OPTIONAL {?album foaf:maker ?maker .
            ?maker foaf:name ?makerName .}
  OPTIONAL {?track dc:title ?trackTitle .}
}
```

Fig. 5. SPARQL query allowing to emphasize the inaccuracy of query 4.

DBpedia dataset Regarding the DBpedia dataset, we express here only a few observations about failed queries. The interpretation of the question "Who is the mayor of Berlin?" does not return any result because the generated SPARQL query used the property `dbo:mayor` instead of `dbo:leader`. Here again SWIP made use of a property which is closer to the expressed user need but unfortunately not used in the data.

The interpretation of "Who is the husband of Amanda Palmer?" does not return any result because it makes use of the property `dbo:spouse` but not the property `dbp:spouse`, which is the only property that give us results for this question.

5 Conclusion and future work

In this paper, we presented the performance of the SWIP approach on the QALD-3 benchmarking task. The results on Musicbrainz are clearly encouraging, those on DBpedia are quite disappointing. It appears that the approach

depends on the quality of the target KB. The data in this KB must be quite homogeneous and respect the framework fixed by the schema it is based on. However, most of the triple stores currently available on the Web of linked data do not fulfill these requirements.

After these observations, we want to explore new leads allowing the approach to evolve and stick more to the data itself than to the ontology. This aspect will definitely be added to the objectives of our current work on automatic/assisted pattern construction. We also intend to improve the matching process and add the support of aggregates in order to obtain better results for the next challenge and improve the user experience.

References

1. Athanasis, N., Christophides, V., Kotzinos, D.: Generating on the fly queries for the semantic web: The ics-forth graphical rql interface (grql). In: International Semantic Web Conference. pp. 486–501 (2004)
2. Cabrio, E., Cojan, J., Aprosio, A., Magnini, B., Lavelli, A., Gandon, F.: Qakis: an open domain qa system based on relational patterns. In: International Semantic Web Conference (Posters & Demos) (2012)
3. Clemmer, A., Davies, S.: Smeagol: a "specific-to-general" semantic web query interface paradigm for novices. In: Proceedings of the 22nd international conference on Database and expert systems applications - Volume Part I. pp. 288–302. DEXA'11, Springer-Verlag, Berlin, Heidelberg (2011), `http://dl.acm.org/citation.cfm?id=2035368.2035396`
4. CoGui: A conceptual graph editor. Web site (2009), http://www.lirmm.fr/cogui/
5. Elbassuoni, S., Ramanath, M., Schenkel, R., Weikum, G.: Searching rdf graphs with sparql and keywords. IEEE Data Eng. Bull. 33(1), 16–24 (2010)
6. Ferré, S., Hermann, A.: Reconciling faceted search and query languages for the semantic web. International Journal of Metadata, Semantics and Ontologies 7(1), 37–54 (2012)
7. Ferré, S.: Squall: A controlled natural language for querying and updating rdf graphs. In: Controlled Natural Language, pp. 11–25. Springer (2012)
8. Kaufmann, E., Bernstein, A.: Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. Web Semantics: Science, Services and Agents on the World Wide Web 8(4), 377–393 (2010)
9. Lehmann, J., Bühmann, L.: Autosparql: let users query your knowledge base. In: Proceedings of the 8th extended semantic web conference on The semantic web: research and applications-Volume Part I. pp. 63–79. Springer-Verlag (2011)
10. Lei, Y., Uren, V.S., Motta, E.: Semsearch: A search engine for the semantic web. In: EKAW. pp. 238–245 (2006)
11. Nivre, J., Hall, J., Nilsson, J.: Maltparser: A data-driven parser-generator for dependency parsing. In: Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006), May 24-26, 2006, Genoa, Italy. pp. 2216–2219. European Language Resource Association, Paris (2006)
12. Pradel, C., Haemmerlé, O., Hernandez, N.: Expressing conceptual graph queries from patterns: how to take into account the relations. In: Proceedings of the 19th International Conference on Conceptual Structures, ICCS'11, Lecture Notes in Artificial Intelligence # 6828. pp. 234–247. Springer, Derby, GB (July 2011)

13. Pradel, C., Haemmerlé, O., Hernandez, N.: A semantic web interface using patterns: The swip system (regular paper). In: Croitoru, M., Rudolph, S., Wilson, N., Howse, J., Corby, O. (eds.) IJCAI-GKR Workshop, Barcelona, Spain, 16/07/2011-16/07/2011. pp. 172–187. No. 7205 in LNAI, Springer, http://www.springerlink.com (mai 2012)
14. Raimond, Y., Abdallah, S., Sandler, M., Giasson, F.: The music ontology. (2007)
15. Russell, A., Smart, P.R.: Nitelight: A graphical editor for sparql queries. In: International Semantic Web Conference (Posters & Demos) (2008)
16. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In: ICDE. pp. 405–416 (2009)
17. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A., Gerber, D., Cimiano, P.: Template-based question answering over rdf data. In: Proceedings of the 21st international conference on World Wide Web. pp. 639–648. ACM (2012)
18. Wang, H., Zhang, K., Liu, Q., Tran, T., Yu, Y.: Q2semantic: a lightweight keyword interface to semantic search. In: Proceedings of the 5th European semantic web conference on The semantic web: research and applications. pp. 584–598. Springer-Verlag (2008)
19. Zhou, Q., Wang, C., Xiong, M., Wang, H., Yu, Y.: Spark: Adapting keyword query to semantic search. In: ISWC/ASWC. pp. 694–707 (2007)